CURSO DE GIT

En esta sección #1 del curso aprenderás:

Introducción a Git

Git es un sistema de control de versiones. Un sistema de control de versiones nos va a servir para trabajar en equipo de una manera mucho más simple y optima cuando estamos desarrollando software. ... Cuando acabamos de desarrollar nuestro código, utilizamos Git para mezclar los cambios con los otros compañeros.

Los sistemas de **control de versiones** son una categoría de herramientas de software que ayudan a un equipo de software a gestionar los cambios en el código fuente a lo largo del tiempo.

Un **control de versiones** es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar **versiones** específicas más adelante.

Instalando Git

Antes de empezar a utilizar Git, tienes que instalarlo en tu computadora.

Ubuntu:

Correr esto en la terminal

apt-get install git

Para otras distribuciones de linux:

https://git-scm.com/download/linux

Mac:

Correr esto en la terminal

brew install git

Si no tienes instalado brew, corre esto en la terminal

/usr/bin/ruby -e "$(curl -fsSL <https://raw.githubusercontent.com/Homebrew/install/master/install>)"

Windows:

https://git-scm.com/download/winfinalmentegit --version

Para confirmar que has instalado bien git, confirma que este comando te arroje la version de git que tienes instalado

git --version

Configuración Inicial

Esto se hace una sola vez en la computadora.

Puedes volver a cambiarla ejecutando los comandos correspondientes.

git config --global user.name “NOMRE DEL USUARIO”

git config --global user.email “CORREO DEL USUARIO”

Para confirmar los datos

git config --list

git config user.name

git config user.email

Comandos básicos

git init > inicializa un repositorio de git > ls -a > .git > cd .git : dentro de la carpeta > salir de la carpeta .git > .. cd

git add > del working directory al stating > git add nombre del archive o

git add . > git add --all para guardar todo de una

git status > vemos en que estado estan los archivos

git commit -m “comentario” para mandar del stating > al repositorio local

Para modificar en el bin

git commit > comentario arriba > salir con ESC :wq ENTER

git log para ver quien cuando ejecutaron los commit > información de los commit > solo se pueden ver los commit de la rama en la que estas.

git remote add origin y la url> es para subir nuestro proyecto local remotamente

ejemplo: git remote add origin <https://github.com/alejandrosergio/curso-de-git.git>

git remote -v > para mirar el repositorio remoto

git push origin y nombre de la rama > eviar archivos del repositorio local al remoto > origin el origen al que va llegar en el repositorio remoto o

upstream que se utilizan por convención.

git pull origin y el nombre de la rama > Traer cambios del repositorio remoto > súbalo con push para yo traerlo con pull.

git help para resivir una ayuda de los comandos básicos de git

git help --all ver todos los comandos de git

gir help nombre de comando recibir documentación de ese comando en la web

ELIMINAR En esta sección #2 del curso aprenderás:

Como regresar a etapas anteriores en:

* + - * + Working Directory
        + Stage
        + Commit

git clean AND git checkout

git clean Remueve los archivos no traqueados del working Dyrectory.

git clean -f Fuerza a git a remover los archivos.

git clean -n Miras que archivos vas a eliminar

touch y “nombre de archivo” Para crear un archivo en Linux desde el terminal

git checkout Elimina los cambios de los archivos traqueados > estado modificado

git reset

git reset HEAD “Nombre del archivo” > elimina la HEAD “cabezera” > resetea el head > elimina del stage y lo pasa al working Dyrectory

git revert

git revert y “nombre del comig hexadecimal” > git log > buscamoes el commit > copiamos su numero único > y ejecutamos el comando revierte hasta ese commit, elimina todos los que estaban después de el junto a el.

git mv

git mv “nombre archivo viejo con la extencion” “nombre archivo nuevo con la extencion” > hacer el commit -m “comentario ” > guardar los cambios

que hace vm

mv nombre\_viejo nombre\_nuevo

git add nombre\_nuevo

git rm nombre\_viejo

En esta sección #3 del curso aprenderás:

* Algoritmos para hacer merge
* Resolucion de conflictos
* "Viaje en el tiempo"
* Stash
* Versionar tu codigo

git branch

git branch Lista las ramas actuales de en tu repositorio local

la que tiene un \* es la rama en la que estas posicionado.

git branch “nombre de la rama” Para crear una nueva rama.

git branch -d “nombre de la rama” Para eliminar una rama

git branch -m “nombre\_viejo” “nombre\_nuevo” cambiar nombre de una rama

git checkout

git checkout y el hexadecimal de el commit nos permite ir al código de ese commit sin eliminar el codigo anterior y sus commit’s > podemos volver al actual también.

git checkout “nombre de la rama” nos permite ir al ultimo commit de la rama

le dice al HEAD que apunte al ultimo commit

git branch -b “nombre de la rama” nos permite crear y pasarnos a una rama nueva

git merge

git merge “nombre de la rama a unir” nos sirve para unir código de dos ramas > se combina el código de la rama llamada con la rama donde se ejecuta el comando // se une el código y la historia de git (commit’s)

Fast – forward || Recursive > strategy

git stash

git stash > nos permite guardar un código incompleto para ir a otra rama sin perder los cambios ni mandar el archivo al stegin área.

git stash pop > nos devuelve los datos guardados en el stash.

NOTA: Li-Fo > last in – first out

Los estash solo funciona en los documento que estas editando en su momento > por eso debes hacer stash una vez > si lo haces y luego modificas otro sin hacer el stash pop > tendras que hacer otro stash y este se apilara > los apilas > el ultimo stash es el primer stash pop que aparece por la pila > no puedes sacar de una pirámide el bloque de abajo sin lo que de arriba caiga > vas apilando y luego vas sacando desde arriba que es lo ultimo guardado en el stash

git stash list > Nos lista las posiciones guardadas en la rama

git stash clear > Nos elimina las posiciones donde se guarda el stash > elimina los cambios guardados del stash

git tag > es un apuntador a un commit > se pueden crear tag de commit anteriores

git tag “nombre del tag” > tag ligero > poca metadata

git tag -a “nombre del tag” y luego un comentario -m “comentario”

> tag anotado > mas metadata

git tag > lista los tag’s

git checkout “nombre del tag” viajar de tag a tag

git tag -d “nombre del tag” elimina un tag

NOTA: se reomienda usar los tag’s para el versionamiento.

En esta sección #4 del curso aprenderás:

* Interaccion con repositorios remotos:
* Registrar repositorio remoto
* Enviar informacion
* Recibir información

git remote

git remote > nos lista la cantidad de repositorios remotos que tenemos agregados actualmente

git remote -v > nos lista la cantidad de repositorios remotos que tenemos agregados actualmente pero con una mayor información

git remore add “nombre por convencion” y URL > encontramos por convención 2 terminos > origin y upstream > esto nos sirve para agregar un repositorio local a un repositorio remoto ya sea en GitHub o GitLab

git clone

git clone por https

* git clone y url de github > nos clona idénticamente el repositorio carpetas y archivos iguales

git clone por SSH > PASO A PASO

* ls ~/.ssh > listar para verificar si tenemos ssh key
* ssh-keygen -t rsa -b 4096 -C “email” > crear una clave ssh
* donde quieres que sea guardado ENTER
* contraseña ENTER si no quieres password
* ENTER
* ssh/id\_rsa > .pub el archivo que queremos
* cd ~ ir al home del pc
* code .ssh/id\_rsa.pub nos abre la clave en el editor VS
* Copiamos la clave del archivo que se abrió
* En GitHub > settings > SSH and GPG Keys > new ssh key > elegimos un titulo y pegamos lo copiado > add SSH key
* Copiamos el enlace que nos da le reposiorio que queremos clonar con SSH
* git clone “url” clonamos el repositorio con SSH
* Terminaste!!!

Ejecutar vscode en terminal

-Abran su Visual Studio Code

-Abran en el Menu la pestaña "Vista"

-Abran la paleta de comandos (Control + Shift + P)

-Escriban shell en la terminal de comandos que aparece

-Denle click en Shell Command: Install 'code' command in PATH command

git fetch =! de git pull > pero parecido

git fetch (origin || upstream) (master || otra rama) > atraemos los cambios y no los mezclamos como con git pull

git checkout (origin || upstream) / (master || otra rama) > Podemos ver los cambios implementados en nuestro código

git merge (origin || upstream) / (master || otra rama) > unimos los cambios del fetch a nuestro código

git pull

git pull (origin || upstream) (master || otra rama) > atraemos los cambios y los mezclamos

git push

git push (origin || upstream) (master || otra rama) > Enviamos los cambios (archivos y historias) de nuestro repositorio local a nuestro repositorio remoto

CUANDO DA ERRORES

* git push -force (origin || upstream) (master || otra rama) > forsamos hacer el push pero elimina los commit que hay en el repositorio remoto

LA MEJOR FORMA

* git pull (origin || upstream) (master || otra rama) > atraemos los cambios y solucionamos conflictos si los hay

* git push (origin || upstream) (master || otra rama) > Enviamos los cambios (archivos y historias) de nuestro repositorio local a nuestro repositorio remoto

En esta sección #5 del curso aprenderás:

* Ver información especifica acerca ciertos objetos
* Comparación

git show > mostrar información de los objetos > los tagas o commits

mostrara metadata y cambios de contenido de un commit especifico

Commit

git show “nombre de un commit hexadecimal” > nos permite ver el commit y los cambios que se guardaron en el commit.

Tag

git show “nombre del tag > nos permite ver el tag y su apuntador > commit con los cambios que se guardaron en el commit.

git diff una función que toma como parámetro dos fuentes de datos y nos retorna los cambios entre dichas fuentes de datos > commit ramas archivos ETC. > se utiliza con git status y git log para analizar el estado actual del repositorio.

git diff “nombre del commit hexadecimal” “nombre del commit hexadecimal” > nos muestra las diferencias > los cambios que hubo entre esos dos commit. > si pasamos un único parámetro lo compara con el ultimo commit echo.

git diff > sin aver echo el commit nos muestra los cambios que hicimos en el archivo

git shortlog > resume el output del commando git log > otra version > agrupa los

commig por autor > y solo muestra el nombre del commit.

git shortlog > muestra el nombre del autor con la cantidad total de commit eschos por el (N) y lista el comentario de todos esos commit.

git shortlog --summary --numbered > nos muestra el numero de commit

echos por cada autor.

git describe > Busca el tag mas reciente desde un commit > solo muestra tag anotados

git describe > nos muestra el tag, los commit que pasaron por el tag

y su identificador único > si no ah pasado por commit solo muestra su nombre.

EN esta sección #6 del curso aprenderás:

* 2 métodos para hacer reparación avanzada

git cherry-pick > seleccionar una cereza > aplicar los cambios introduccidos por un commit que allas elegido > un parche al commit > enlazar > uno o mas commit > aplicara el cambio que se introduce en los commit > también se creara un commit por cada uno de ellos.

git cherry-pick “nombre hexadecimal del commit” > trae un commit de una rama a otra > y agrega ese código

git rebase > se usa mas cuando trabajas solo y no has hecho push > lo que hace es crear una historia lineal

git rebase “nombre de la rama” > lo de arriba la verdad no se aconseja usar a menos que este trabajando solo en dos ramas distintas y quieras unir solo una

rama con los commit de dos ramas unidas a una > para una historial lineal

En esta sección #7 del curso aprenderás:

* Conjunto de herramientas extras
* Conceptos extra interesantes

git blame > culpar > revisemos quien hizo una modificación en el proyecto.

PASO A PASO

* pwd > miramos la dirección de donde estamos parados es necesario que al ejecutar el git blame el archivo este en el mismo nivel.
* ls > mirar que el archivo al que queremos ejecutar el git blame este en la lista
* git blame “nombre del archivo” > nos enseña el hexadecimal de un commit el autor y en que fecha se hizo
* git blame -L 3,5 “nombre del archivo” > nos enseña las líneas que queremos ver con la información de arriba

git grep > imprime líneas que encajen con un patrón.

git grep “patron” > nos imprime el archivo y el código junto al patrón resaltado de rojo de lo que que quisimos buscar.

git grep -n “patron” > nos imprime el archivo, numero de linea y el código junto al patrón resaltado de rojo de lo que que quisimos buscar.

git grep -c “patron” > nos imprime cuantas veces aparece el patrón en cada archivo.

git grep “patron” “commit hexadecimal” > nos imprime el nombre del commmit, archivo y el patrón.

HEAD~ y HEAD^ > hacemos referencias a commit con respecto al head y nos movemos a el

git checkout HEAD~”numero que queremos referenciar decendente padre del padre” > como un iterador que incrementa por cada commit 1 2 3 voy bajando la referencia del commit.

git checkout HEAD^”numero que queremos referenciar decendente padre del padre” >

git ignore > se especifican archivos que no queremos que sean traqueados en nuestros proyecto.

PASO A PASO

* Creamos un archivo llamado .gitignore
* Para crear un comentario se hace con # minimo 1
* Simplemente colocas la ruta con / para entrar y llegar al nivel
* Podemos utilizar \*.txt > ejemplo para decir todos

git alias > creemos nuestros propios comandos

git config --global alias.”nombre que queremos poner al comando” ‘entre comillas colocas el comando original’ > creamos el comando